Lecture 8: Reasoning with Inconsistent Knowledge

2-AIN-144/2-IKV-131 Knowledge Reperesentation & Reasoning

Martin Baláž, Martin Homola

Department of Applied Informatics
Faculty of Mathematics, Physics and Informatics
Comenius University in Bratislava

18 Apr 2013

Reasoning with Inconsistent Knowledge

1. Explosive Approach
   The whole language is the only meaning of a contradictory knowledge base (falsity implies anything).
2. Paraconsistent Approach
   Accept contradictory information and perform reasoning tasks that take it into account.
3. Update/Belief Revision Approach
   Update/revise the knowledge base in order to regain consistency.

Example

$$\underbrace{\mathit{train} \qquad\qquad \rlap{$\overbrace{\emptyset \qquad\qquad \neg\,\mathit{train}}^{\mathit{not}\ \mathit{train}}$}}_{\mathit{not}\ \neg\,\mathit{train}} \qquad\qquad\qquad\qquad$$

$$\begin{aligned}
\mathit{cross} &\leftarrow \neg\,\mathit{train}\\
\neg\,\mathit{cross} &\leftarrow \mathit{not}\ \neg\,\mathit{train}\\
\mathit{listen} &\leftarrow \mathit{not}\ \mathit{train}, \mathit{not}\ \neg\,\mathit{train}
\end{aligned}$$

Syntax of Extended Logic Programs

Definition (Literal)

A *classical literal* is an atom A or a classically negated atom $\neg A$. A *default literal* is a default negated classical literal *not* L. A *literal* is either a classical literal L or a default literal *not* L.

Definition (Extended Logic Program)

An *extended logic program* is a set P of rules

$$L_0 \leftarrow L_1, \dots, L_m, not\ L_{m+1}, \dots, not\ L_n$$

where $0 \leq n$ and $L_0, \dots, L_n$ are classical literals.

Definition (Interpretation)

An *interpretation* is a subset I of the extended Herbrand base $\mathcal{B}^{\neg} = \mathcal{B} \cup \{\neg A \mid A \in \mathcal{B}\}$.

Given an interpretation I and an atom $A \in \mathcal{B}$,

- A is *true* iff $A \in I$ and $\neg A \notin I$
- A is *false* iff $A \notin I$ and $\neg A \in I$
- A is *unknown* iff $A \notin I$ and $\neg A \notin I$
- A is *inconsistent* iff $A \in I$ and $\neg A \in I$

Given an interpretation I and a classical literal $L \in \mathcal{B}^{\neg}$,

- $I \models L$ iff $L \in I$
- $I \models \textit{not}\ L$ iff $L \notin I$

An interpretation I is *consistent* iff $\{A, \neg A\} \not\subseteq I$ for any $A \in \mathcal{B}$.
An extended logic program is *consistent* iff it has a model.

Stable Model (Answer Set)

Definition (Program Reduct)

Let P be a normal logic program and I be an interpretation. The reduct of P (with respect to I) is a positive logic program P^I obtained from P by

- removing rules containing a default literal L in the body such that $I \not\models L$
- removing remaining default literals L, i.e. default literals with $I \models L$

Definition (Stable Model)

An interpretation I is a *stable model* of an extended logic program P if

- P^I is consistent and I is the least model of P^I, or
- P^I is inconsistent and $I = \mathcal{B}^\neg$.

Properties of Stable Models

Proposition

Let P be an extended logic program. Then P has an inconsistent stable model iff P is inconsistent.

Proposition

Let P be an extended logic program. If P is inconsistent then $\mathcal{B}^{\neg}$ is the only stable model of P.

Proposition

Stable models of an extended logic program are minimal models.

Example

The interpretation $I = \{a\}$ is a minimal model of the extended logic program $P = \{a \leftarrow \mathit{not}\, a\}$, but I is not a stable model of P.

Birds usually fly (except penguins, ostriches, birds with boken wings, ...).

$$
\begin{array}{rcl}
fly(X) & \leftarrow & bird(X), not\ ab(X) \\
ab(X) & \leftarrow & penguin(X) \\
ab(X) & \leftarrow & ostrich(X) \\
ab(X) & \leftarrow & bird(X), broken\_wing(X)
\end{array}
$$

$$
\begin{array}{rcl}
fly(X) & \leftarrow & bird(X), not\ \neg fly(X) \\
\neg fly(X) & \leftarrow & penguin(X) \\
\neg fly(X) & \leftarrow & ostrich(X) \\
\neg fly(X) & \leftarrow & bird(X), broken\_wing(X)
\end{array}
$$

Closed World Assumption

$$
\begin{aligned}
person(peter) &\leftarrow \\
person(bob) &\leftarrow \\
person(alice) &\leftarrow \\
\\
employee(peter) &\leftarrow \\
employee(bob) &\leftarrow \\
\\
\neg\, employee(X) &\leftarrow \quad person(X),\ not\ employee(X)
\end{aligned}
$$

Open World Assumption

$$
\begin{aligned}
person(peter) &\leftarrow \\
person(bob) &\leftarrow \\
person(alice) &\leftarrow \\
\\
employee(peter) &\leftarrow \\
\neg\, employee(bob) &\leftarrow
\end{aligned}
$$

$$
\begin{aligned}
person(peter) &\leftarrow \\
person(bob) &\leftarrow \\
person(alice) &\leftarrow \\
\\
employee(peter) &\leftarrow \\
employee(bob) &\leftarrow \\
\\
\neg\, employee(X) &\leftarrow \quad person(X), not\ employee(X) \\
employee(X) &\leftarrow \quad person(X), not\ \neg\, employee(X)
\end{aligned}
$$

Consider the following logic program P:

$$\begin{aligned} a &\leftarrow \\ \neg\, a &\leftarrow \\ b &\leftarrow \end{aligned}$$

$I = \{a, \neg\, a, b, \neg\, b\} = \mathcal{B}^{\neg}$ is the only answer set of P (because P is inconsistent).

Although we have contradictory information about a, we could say something reasonable about b.

Paraconsistent Stable Model

Definition (Program Reduct)

Let P be an extended logic program and I be an interpretation. The reduct of P (with respect to I) is a positive logic program P^I obtained from P by

- removing rules containing a default literal L in the body such that $I \not\models L$
- removing remaining default literals L, i.e. default literals with $I \models L$

Definition (Paraconsistent Stable Model)

An interpretation I is a *paraconsistent stable model* (abbreviately p-stable model) of an extended logic program P if I is the least model of P^I.

Properties of Paraconsistent Stable Models

Proposition

Consistent stable models are paraconsistent stable models.

Example

The interpretation $I = \{a, \neg a, b\}$ is a paraconsistent stable model of the extended logic program $P = \{a \leftarrow; \neg a \leftarrow; b \leftarrow\}$, but I is neither consistent nor a stable model of P.

Proposition

Paraconsistent stable models are minimal models.

Example

The interpretation $I = \{a\}$ is a minimal model of the extended logic program $P = \{a \leftarrow \mathit{not}\, a\}$, but I is not a paraconsistent stable model of P.

Stable Models vs. Paraconsistent Stable Models

Example

$$P = \left\{ \begin{array}{rcl} \neg\, a & \leftarrow & \\ a & \leftarrow & \mathit{not}\ b \end{array} \right\}$$

has the p-stable model $\{a, \neg\, a\}$, while it has no stable model.

Example

$$P = \left\{ \begin{array}{rcl} a & \leftarrow & \\ \neg\, a & \leftarrow & \\ b & \leftarrow & \mathit{not}\ b \end{array} \right\}$$

has the stable model $\mathcal{B}^{\neg}$, while it has no p-stable model.

Update/Belief Revision Approach

$$P_1 = \left\{ \begin{array}{rcl} sleep & \leftarrow & not\ tv\_on \\ watch\_tv & \leftarrow & tv\_on \\ tv\_on & \leftarrow & \end{array} \right\}$$

$$P_2 = \left\{ \begin{array}{rcl} \neg\, tv\_on & \leftarrow & power\_failure \\ power\_failure & \leftarrow & \end{array} \right\}$$

$$P_3 = \left\{ \neg\, power\_failure \leftarrow \right\}$$

Definition (Dynamic Logic Program)

A *dynamic logic program* is a non-empty sequence of extended logic programs $\mathcal{P} = P_1 \oplus P_2 \oplus \cdots \oplus P_n$.

Definition (Dynamic Jusitified Update)

An interpretation I is a *dynamic justified update* of a dynamic logic program $\mathcal{P} = P_1 \oplus P_2 \oplus \cdots \oplus P_n$ if I is a stable model of $\mathit{Residue}(\mathcal{P}, I)$ where

$$\begin{aligned} \mathit{Reject}(\mathcal{P}, I, i) &= \{r \in P_i \mid \exists r' \in P_j : i < j, I \models \mathit{body}(r'), \\ &\qquad\qquad \mathit{head}(r') = \neg\, \mathit{head}(r)\} \\ \mathit{Residue}(\mathcal{P}, I) &= \bigcup_{1 \leq i \leq n} [P_i \setminus \mathit{Reject}(\mathcal{P}, I, i)] \end{aligned}$$

Example

$$P_1 = \left\{ \begin{array}{rcl} sleep & \leftarrow & not\ tv\_on \\ watch\_tv & \leftarrow & tv\_on \\ tv\_on & \leftarrow & \end{array} \right\}$$

$$I_1 = \{tv\_on, watch\_tv\}$$

$$\begin{array}{rcl} Reject(P_1, I_1, 1) & = & \emptyset \\ Residue(P_1, I_1) & = & P_1 \end{array}$$

Example

$$P_1 = \left\{ \begin{array}{rcl} sleep & \leftarrow & not\ tv\_on \\ watch\_tv & \leftarrow & tv\_on \\ tv\_on & \leftarrow & \end{array} \right\}$$

$$P_2 = \left\{ \begin{array}{rcl} \neg\, tv\_on & \leftarrow & power\_failure \\ power\_failure & \leftarrow & \end{array} \right\}$$

$$I_2 = \{power\_failure, \neg\, tv\_on, sleep\}$$

$$\begin{array}{rcl} Reject(P_1 \oplus P_2, I_2, 2) & = & \emptyset \\ Reject(P_1 \oplus P_2, I_2, 1) & = & \{tv\_on \leftarrow\} \\ Residue(P_1 \oplus P_2, I_2) & = & (P_1 \setminus \{tv\_on \leftarrow\}) \cup P_2 \end{array}$$

Example

$$P_1 = \left\{ \begin{array}{rcl} sleep & \leftarrow & not\ tv\_on \\ watch\_tv & \leftarrow & tv\_on \\ tv\_on & \leftarrow & \end{array} \right\}$$

$$P_2 = \left\{ \begin{array}{rcl} \neg\, tv\_on & \leftarrow & power\_failure \\ power\_failure & \leftarrow & \end{array} \right\}$$

$$P_3 = \left\{ \begin{array}{rcl} \neg\, power\_failure & \leftarrow & \end{array} \right\}$$

$$I_3 = \{\neg power\_failure, tv\_on, watch\_tv\}$$

$$\begin{array}{rcl} Reject(P_1 \oplus P_2 \oplus P_3, I_3, 3) & = & \emptyset \\ Reject(P_1 \oplus P_2 \oplus P_3, I_3, 2) & = & \{power\_failure \leftarrow\} \\ Reject(P_1 \oplus P_2 \oplus P_3, I_3, 1) & = & \emptyset \\ Residue(P_1 \oplus P_2 \oplus P_3, I_3) & = & P_1 \cup (P_2 \setminus \{power\_failure \leftarrow\}) \cup P_3 \end{array}$$